Programmazione e Architetture degli Elaboratori - Foglio 5

Luca Manzoni, Michele Rispoli, Pietro Morichetti

Esercizio 01

Si scriva un programma C in cui, dato un numero intero `N`, vengono creati `N` thread, ciascuno dei quali stampa a schermo il messaggio `"Hello from thread %d!"`, dove `%d` è il thread-id di quel thread (ottenibile tramite la funzione `pthread_t pthread_self(void)`. Il main thread inoltre stampa a schermo `"Hello from main thread!"`.

Sol:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>


//Funzione eseguita dal thread. La firma deve essere questa
void* helloThread(void* args){

    pthread_t mytid = pthread_self();
    printf("Hello from thread %d!\n",mytid);
}

int main(){

    int N= 10;

    for (int iii=0;iii<N;++iii){
        pthread_t t;
        pthread_create(&t, NULL, helloThread, NULL);
    }

    printf("Hello from Main thread!\n");

    return 0;
}
```

Esercizio 02

Si scriva un programma C in cui, dato un numero intero `N`, viene creato un array `A` di interi, da 1 a N, dopodiché viene creato un thread, che dovrà calcolare la media dei valori contenuti in `A` e salvare il valore in un float. Il thread che calcola la media deve ricevere in input sia `A`,`N` e l'indirizzo dove scrivere il risultato. Una volta terminato il calcolo il main thread deve stampare a schermo il valore della media. (Suggerimento: per passare più argomenti a un thread possiamo utilizzare le `struct`)

Sol:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>


//Struct usato per passare gli argomenti al thread
struct Args{
    int* A;
    int* size;
    float* out;
};

//Funzione eseguita dal thread. La firma deve essere questa
void* media(void* args){

    int* A = ((struct Args*)args)->A;
    int* size= ((struct Args*)args)->size;
    float* out= ((struct Args*)args)->out;

    float sum = 0;
    int count = 0;

    for (int iii=0;iii<*size;++iii){
        sum += A[iii];
        ++count;
    }
    *out = sum / count;
}

int main(){
```

```
    int N= 10;

    //inizializzo A
    int A[N];
    for (int iii=0;iii<N;++iii){
        A[iii]=iii+1;
    }

    float mean=0;

    struct Args* args = (struct Args*) malloc(sizeof(struct Args*));
    args->A = A;
    args->size = &N;
    args->out = &mean;

    pthread_t t;
    pthread_create(&t, NULL, media, args);
    pthread_join(t, NULL);

    printf("Main thread: mean is %.2f\n",mean);

    free(args);

    return 0;
}
```

Esercizio 03

Implementare l'algoritmo mergeSort (link alla pagina wikipedia) in maniera che le chiamate ricorsive vengano eseguite da dei nuovi thread a ogni passo. Una possibile implementazione dell'algoritmo può essere costituita da metodi con le seguenti signature:

```
void merge(int* A, int N);
void* mergeSort(void* args);
```

dove l'unico argomento della routine `mergeSort` potrebbe essere lo `struct`

```
// arg pack
struct Args{
    int* A;
    int N;
};
```

Per testare l'algoritmo potete inizializzare un array ordinato come al solito e darlo poi in pasto alla seguente routine che lo rimescola:

```
// array shuffling utility
void naiveShuffle(int*A,int N){
    srand(time(NULL));
    for (int iii=0;iii<N;++iii){
        int jjj;
        do{
            jjj=rand()%N;
        } while(iii==jjj);

        int buff = A[iii];
        A[iii]=A[jjj];
        A[jjj]=buff;
    }
}
```

Sol:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// array printing utility
void printarr(int*A,int N){
    for (int iii=0;iii<N;++iii){
        printf("%d ",A[iii]);
    }
    printf("\n");
}

// array shuffling utility
void naiveShuffle(int*A,int N){
    srand(time(NULL));
    for (int iii=0;iii<N;++iii){
        int jjj;
        do{
            jjj=rand()%N;
        } while(iii==jjj);

        int buff = A[iii];
        A[iii]=A[jjj];
        A[jjj]=buff;
```

```
    }
}

// arg pack
struct Args{
    int* A;
    int N;
};

void merge(int* A, int N){

    // Copiati A
    int* buffer = (int*)malloc(sizeof(int)*N);
    for(int iii=0; iii<N;++iii){
        buffer[iii]=A[iii];
    }

    // Puntatori al primo e ultimo elemento delle due metà di A
    int *l = buffer,    *lf = buffer + N/2;
    int *r = lf,        *rf = buffer + N;
    while(1){
        if(l<lf){
            if(r<rf && *r<*l){
                *A=*r;
                ++r;
            }
            else{
                *A=*l;
                ++l;
            }
        }
        else if(r<rf){
            *A=*r;
            ++r;
        }
        else{
            break;
        }
        ++A;
    }
    free(buffer);
}

void* mergeSort(void* args){
```

```
    //unpack args
    int* A = ((struct Args*)args)->A;
    int N  = ((struct Args*)args)->N;

    //base case
    if(N<2){
        return NULL;
    }

    //recursive case
    pthread_t tl, tr;
    struct Args argl;
    argl.A = A;
    argl.N = N/2;
    pthread_create(&tl,NULL,mergeSort,&argl);

    struct Args argr;
    argr.A = A+N/2;
    argr.N = N/2+N%2;
    pthread_create(&tr,NULL,mergeSort,&argr);

    //wait before merging
    pthread_join(tl,NULL);
    pthread_join(tr,NULL);
    merge(A,N);
    printarr(A,N);

}

int main(){
    int N=10;

    int* A = (int*)malloc(sizeof(int)*N);;
    for (int iii=0;iii<N;++iii){
        A[iii]=iii+1;
    }

    //shuffle array
    naiveShuffle(A,N);
    printarr(A,N);

    struct Args args;
    args.A = A;
```

```
    args.N = N;
    mergeSort(&args);

    free(A);
    return 0;
}
```